**Лабораторная работа № 2 “Построение выпуклой оболочки по алгоритму Грэхема”.**

**ЦЕЛЬ**

Разобраться в алгоритме построения выпуклой оболочки по алгоритму Грэхема и запрограммировать его.

**РЕАЛИЗАЦИЯ**

Был реализован алгоритм на языке Java в соответствии с псевдокодом из презентации. Пользовательский интерфейс построен так: пользователь вводит количество точек, которые генерируются случайным образом, и задержку в миллисекундах. Далее реализована пошаговая реализация алгоритма с задержкой, установленной пользователем. В любой момент выполнение программы можно остановить.

**ИСПОЛЬЗОВАННЫЕ СТРУКТУРЫ ДАННЫХ**

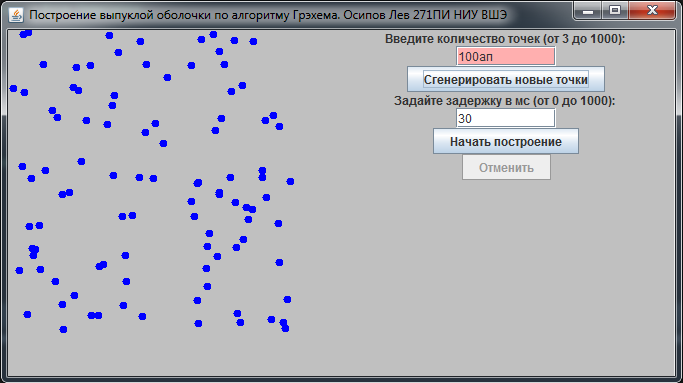
* Класс **Task2** – класс с методом main, который является точкой входа в программу.
* Класс **Point** – класс, моделирующий точку на плоскости. Содержит координаты X и Y, а также позволяет «вычитать» одну точку из другой (вычитая их координаты) – это нужно для нахождения координат вектора.
* Класс **Vector** – класс, моделирующий вектор. Содержит конструктор, который принимает точки начала и конца вектора и вычитает их, получая координаты вектора. Также в классе описана функция векторного произведения и функция нахождения длины вектора.
* Класс **StackG** – класс, моделирующий стек. Отличается от обычного стека наличием функции next\_to\_top(), которая возвращает предпоследний объект в стеке.
* Класс **GrahamAlgorithm** – класс, содержащий в себе основные методы для работы алгоритма (случайную генерацию точек, нахождение самой нижней (и левой, если их несколько) точки, сортировки по точек по полярному углу, вычленение точек с одинаковыми полярными углами, а также непосредственного сам алгоритм Грэхема).
* Класс **DrawPanel** – класс, моделирующий визуализацию алгоритма (рисующий на панели точки и линии для пошагового отображения).
* Класс **CFrame** – класс, реализующий элементы пользовательского интерфейса, обработку данных и запуск алгоритма и его визуализации.

**ОЦЕНКА АЛГОРИТМА**

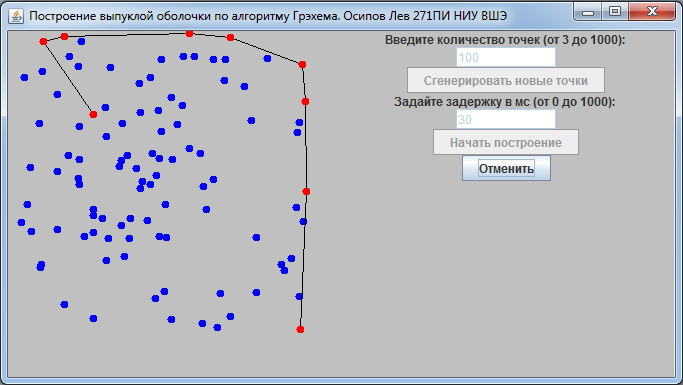
Оценкой времени выполнения алгоритма является О(n\*log(n)). Это обусловлено тем, что большую часть времени выполнения занимает сортировка точек по полярном углу. Так как в программе использована встроенная сортировка Timsort (она является гибридом сортировки вставками и сортировки слиянием), сложность которой является О(n\*log(n)), сложность алгоритма Грэхема для построения выпуклой оболочки будет такой же.

**СКРИНШОТЫ**

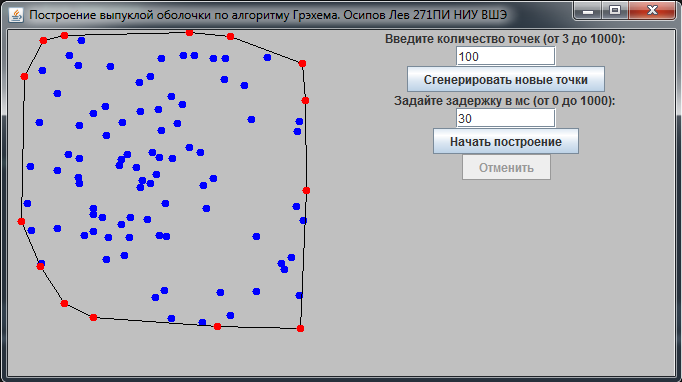
Скриншот ввода данных:



Скриншот процесса выполнения алгоритма:



Скриншот окончания работы алгоритма:



**ВЫВОДЫ**

Был разобран алгоритм Грэхема построения выпуклой оболочки, который оказался весьма интересным. В очередной раз была обнаружена полезность использования векторного произведения. Нахождение выпуклой оболочки является актуальной проблемой, так как используется в различных прикладных задачах.

**СПИСОК ИСТОЧНИКОВ**

Презентация «Вычислительная геометрия. Лекция 2. Построение выпуклой оболочки», ВШЭ.

**ПРИЛОЖЕНИЕ (код программы)**

**Код файла Task2.java**

public class Task2 {

public static void main(String[] args) {

/\*\* Форма и панель \*/

CFrame f = new CFrame();

f.setVisible(true);

f.setResizable(false);

f.setTitle("Построение выпуклой оболочки по алгоритму Грэхема. Осипов Лев 271ПИ НИУ ВШЭ");

}

}

**Код файла Point.java**

public class Point {

private double x;

private double y;

public Point(double x, double y) {

this.x = x;

this.y = y;

}

public double getX() {

return x;

}

public double getY() {

return y;

}

public Point minus(Point op) {

return new Point(op.x - x, op.y - y);

}

@Override

public String toString() {

return "(" + x + " , " + y + ")";

}

}

**Код файла StackG.java**

import java.util.Stack;

/\*\*

\* Моделирует стек.

\*/

public class StackG<E> extends Stack<E>{

public E top() {

return peek();

}

public E next\_to\_top() {

E temp = pop();

E res = peek();

push(temp);

return res;

}

}

**Код файла GrahamAlgoritm.java**

import java.util.\*;

/\*\*

\* Created by Lev on 02.02.14.

\*/

public class GrahamAlgorithm {

/\*\* Самая нижняя (и левая, если их несколько) \*/

private Point p0;

/\*\*

\* Генерирует список точек по их заданному количеству

\* @param count количество

\* @return список точек

\*/

public static ArrayList<Point> generatePoints(int count) {

ArrayList<Point> list = new ArrayList<Point>();

Random gen = new Random();

for (int i = 0; i < count; ++ i) {

list.add(new Point(gen.nextInt(DrawPanel.getSIZE()), gen.nextInt(DrawPanel.getSIZE())));

}

return list;

}

/\*\*

\* Ищет выпуклую оболочку сканированием Грэхема

\* @param Q список точек

\* @return CH(Q)

\*/

public StackG<Point> scanGraham(List<Point> Q, DrawPanel panel, int delay) {

find\_point(Q);

polar\_sort(Q);

check\_for\_same\_angles(Q);

StackG<Point> stack = new StackG<Point>();

stack.push(Q.get(0));

stack.push(Q.get(1));

stack.push(Q.get(2));

panel.setPoints(stack);

for (int i = 3; i < Q.size(); ++ i) {

while (new Vector(stack.next\_to\_top(), stack.top()).composite(new Vector(stack.top(), Q.get(i))) < 0) {

stack.pop();

try {

Thread.sleep(delay);

} catch (InterruptedException e) {

return null;

}

panel.validate();

panel.repaint();

}

stack.push(Q.get(i));

panel.validate();

panel.repaint();

try {

Thread.sleep(delay);

} catch (InterruptedException e) {

return null;

}

}

panel.setEnd(true);

panel.validate();

panel.repaint();

return stack;

}

/\*\*

\* Сортировка стека в порядке возрастания полярного угла,

\* измеряемого против часовой стрелки относительно первой точки.

\* Первая точка должна быть уже определена, она вытаскивается на время сортировки

\* @param src список точек

\*/

public void polar\_sort(List<Point> src) {

src.remove(0);

Collections.sort(src, new Comparator<Point>() {

@Override

public int compare(Point o1, Point o2) {

Vector v1 = new Vector(p0, o1);

Vector v2 = new Vector(p0, o2);

if (v1.composite(v2) > 0) {

return -1;

}

else {

if (v1.composite(v2) < 0) {

return 1;

}

else {

return 0;

}

}

}

});

src.add(0, p0);

}

/\*\*

\* При совпадениях углов точки ближе к p0 удаляются.

\* Первая точка должна быть уже определена.

\* @param src список точек

\*/

public void check\_for\_same\_angles(List<Point> src) {

for (int i = 1; i < src.size() - 1;) {

Vector v1 = new Vector(p0, src.get(i)); // вектор p0pi-1

Vector v2 = new Vector(p0, src.get(i + 1)); // вектор p0pi

if (v1.composite(v2) == 0) {

if (v1.length() < v2.length()) {

src.remove(i);

}

else {

src.remove(i + 1);

}

continue;

}

i++;

}

}

/\*\*

\* Поиск самой нижней точки (если таких несколько, то самой левой)

\* и ее становление на первое место (на дно стека)

\* @param src список точек

\*/

public void find\_point(List<Point> src) {

double minX = src.get(0).getX(), minY = src.get(0).getY();

int ind = 0;

for (int i = 0; i < src.size(); ++ i) {

if (src.get(i).getY() < minY || (src.get(i).getY() == minY && src.get(i).getX() < minX)) { // ниже, либо так же, но левее

minX = src.get(i).getX();

minY = src.get(i).getY();

ind = i;

}

}

Point temp = src.get(ind);

src.set(ind, src.get(0));

src.set(0, temp);

p0 = src.get(0);

}

}

/\*\*

\* Моделирует вектор.

\*/

class Vector {

private Point point; // координаты вектора

public Vector(Point start, Point end) {

point = start.minus(end);

}

public double composite(Vector op) {

return point.getX() \* op.point.getY() - point.getY() \* op.point.getX();

}

public double length() {

return Math.sqrt(point.getX() \* point.getX() + point.getY() \* point.getY());

}

}

**Код файла DrawPanel.java**

import javax.swing.\*;

import java.awt.\*;

import java.util.ArrayList;

/\*\*

\* Рисование области точек и выпуклой оболочки.

\*/

public class DrawPanel extends JPanel {

/\*\* Размер области для рисования \*/

private static final int SIZE = 300;

DrawPanel() {

setBackground(Color.LIGHT\_GRAY);

}

/\*\* Все точки \*/

private ArrayList<Point> all;

/\*\* Точки оболочки \*/

private StackG<Point> st;

/\*\* Рисовать ли последнюю линию \*/

private boolean end = false;

public void setPoints(StackG<Point> st) {

this.st = st;

}

public void setAllPoints(ArrayList<Point> all) {

this.all = all;

}

public void setEnd(boolean end) {

this.end = end;

}

public static int getSIZE() {

return SIZE;

}

/\*\*

\* Рисует точки и линии в зависимости от состояния стека

\*/

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

g.setColor(Color.BLUE);

if (all != null) {

for (Point p : all) {

g.fillOval((int)p.getX() - 4, SIZE - (int)p.getY() - 4, 8, 8); // координаты не центра

}

}

g.setColor(Color.BLACK);

if (st != null) {

for (int i = 1; i < st.size(); ++ i) {

g.drawLine((int)st.get(i).getX(), SIZE - (int)st.get(i).getY(),

(int)st.get(i - 1).getX(),SIZE - (int)st.get(i - 1).getY()); // отрисвока линий

}

if (end) {

g.drawLine((int)st.get(st.size() - 1).getX(), SIZE - (int)st.get(st.size() - 1).getY(),

(int)st.get(0).getX(),SIZE - (int)st.get(0).getY()); // отрисовка последней линии

}

g.setColor(Color.RED);

for (int i = 0; i < st.size(); ++ i) { // красные точки

g.fillOval((int) st.get(i).getX() - 4, SIZE - (int) st.get(i).getY() - 4, 8, 8); // координаты не центра

}

}

}

}

**Код файла CFrame.java**

/\*\*

\* Created by Lev on 02.02.14.

\*/

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.ArrayList;

/\*\*

\* Форма и кнопки

\*/

public class CFrame extends JFrame {

/\*\* Список точек \*/

private ArrayList<Point> list;

/\*\* Панель отрисовки \*/

DrawPanel p = new DrawPanel();

/\*\* Поток выполнения и отрисовки алгоритма \*/

Thread graham;

final JTextField textCount = new JTextField();

final JButton buttonGen = new JButton("Сгенерировать новые точки");

final JTextField textDelay = new JTextField();

final JButton buttonStart = new JButton("Начать построение");

final JButton buttonEnd = new JButton("Отменить");

public CFrame() {

Toolkit tk = Toolkit.getDefaultToolkit();

Dimension d = tk.getScreenSize();

int w = d.width;

int h = d.height;

setSize(w / 2, h / 2);

setLocation(w / 4, h / 4);

setDefaultCloseOperation(WindowConstants.EXIT\_ON\_CLOSE);

createGUI();

}

void createGUI() {

setLayout(new GridLayout(0, 2));

p = new DrawPanel();

p.setPreferredSize(new Dimension(getWidth() / 2, getHeight()));

add(p);

JPanel panel = new JPanel();

panel.setBackground(Color.LIGHT\_GRAY);

panel.setPreferredSize(new Dimension(this.getWidth() / 2, this.getHeight()));

this.add(panel);

panel.setLayout(new BoxLayout(panel, BoxLayout.Y\_AXIS));

JLabel labelCount = new JLabel("Введите количество точек (от 3 до 1000): ");

labelCount.setAlignmentX(Component.CENTER\_ALIGNMENT);

panel.add(labelCount);

textCount.setMaximumSize(new Dimension(100, 20));

textCount.setAlignmentX(Component.CENTER\_ALIGNMENT);

panel.add(textCount);

/\*\* Кнопка генерации новых точек \*/

buttonGen.setAlignmentX(Component.CENTER\_ALIGNMENT);

buttonGen.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

p.setPoints(null);

p.setEnd(false);

int count;

if (textCount.getBackground() != Color.WHITE) {

textCount.setBackground(Color.WHITE);

}

try {

count = Integer.parseInt(textCount.getText());

if (count < 3 || count > 1000) {

throw new NumberFormatException();

}

}

catch (NumberFormatException ex) {

textCount.setBackground(Color.PINK);

return;

}

list = GrahamAlgorithm.generatePoints(count);

p.setAllPoints(list);

p.validate();

p.repaint();

}

});

panel.add(buttonGen);

JLabel labelDelay = new JLabel("Задайте задержку в мс (от 0 до 1000): ");

labelDelay.setAlignmentX(Component.CENTER\_ALIGNMENT);

panel.add(labelDelay);

textDelay.setMaximumSize(new Dimension(100, 20));

textCount.setAlignmentX(Component.CENTER\_ALIGNMENT);

panel.add(textDelay);

/\*\* Кнопка запуска алгоритма \*/

buttonStart.setAlignmentX(Component.CENTER\_ALIGNMENT);

buttonStart.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

graham = new Thread() {

@Override

public void run() {

buttonStart.setEnabled(false);

textCount.setEnabled(false);

textDelay.setEnabled(false);

buttonGen.setEnabled(false);

buttonEnd.setEnabled(true);

p.setEnd(false);

while (buttonStart.isEnabled()) { }

int delay;

if (textDelay.getBackground() != Color.WHITE) {

textDelay.setBackground(Color.WHITE);

}

try {

delay = Integer.parseInt(textDelay.getText());

if (delay < 0 || delay > 1000) {

throw new NumberFormatException();

}

}

catch (NumberFormatException ex) {

textDelay.setBackground(Color.PINK);

return;

}

GrahamAlgorithm alg = new GrahamAlgorithm();

ArrayList<Point> tempList = new ArrayList<Point>(list);

StackG<Point> sol = alg.scanGraham(tempList, p, delay);

textCount.setEnabled(true);

textDelay.setEnabled(true);

buttonGen.setEnabled(true);

buttonStart.setEnabled(true);

buttonEnd.setEnabled(false);

}

};

graham.start();

}

});

panel.add(buttonStart);

/\*\* Кнопка отмены выполнения \*/

buttonEnd.setAlignmentX(Component.CENTER\_ALIGNMENT);

buttonEnd.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

graham.interrupt();

try {

graham.join();

}

catch(InterruptedException e1) {

e1.printStackTrace();

}

textCount.setEnabled(true);

textDelay.setEnabled(true);

buttonGen.setEnabled(true);

buttonStart.setEnabled(true);

}

});

panel.add(buttonEnd);

/\*\* Изначальные действия \*/

buttonEnd.setEnabled(false);

textCount.setText("100");

textDelay.setText("30");

p.setPoints(null);

p.setEnd(false);

list = GrahamAlgorithm.generatePoints(Integer.parseInt(textCount.getText()));

p.setAllPoints(list);

p.validate();

p.repaint();

}

}